Journal of Equity for Science and Sustainable Development

Vol. 1(1), pp. 56-62, June 2017

http://www.mwu.edu.et
Email: *jessd@mwu.edu.et*

Full Length Research Paper

Application Specific Integrated Circuit Implementation of Various Linear Feedback Shift Register Models

C.Nithiya[1], S.Savarirani[2] and S.Ramasamy[3]

[1]*Post-graduate Student, RMKEC, Email: nithiyalingamvlsi@gmail.com*
[2]*Assistant Professor, RMD Engineering College, Email: savarirani@gmail.com*
[3]*Professor Electrical, College of Electrical and Mechanical Engineering, Addis Ababa Science and Technology University, Kilinto, Addis Ababa, Email: rams@aastu.edu.et*

Received: 16 March, 2017; Accepted: 13 June, 2017; Published: 26 June, 2017

ABSTRACT

In many electronic devices, linear feedback shift register (LFSR) is used for generating pseudo-random numbers, pseudo-noise sequences and fast digital counters. So for high performance applications LFSR should generate efficient sequences. The efficient sequences can be generated in so many methods. The aim of the paper is to compare various LFSR implementations in the standard polynomials. The practical guidelines to choose the optimum LFSR design for pseudo-random generator was provided. The comparison of various LFSR implementations in different polynomial was analyzed. The design was simulated and synthesized using synopsys verilog compiler simulator and cadence register transfer language compiler and cadence encounter tool was used for application specific integrated circuit implementation. On comparing with different LFSR implementation, the Fibonacci model offer significant advantage in terms of minimum power and area.

Keywords: *Additive scrambler, Cyclic Redundancy Code, Fibonacci, Galois, Linear Feedback Shift Register, Multiplicative Scrambler.*

Corresponding Author *Address: Email:* savarirani@gmail.com

Introduction

An LFSR is a shift register whose input bit is driven by the XOR of some bits of the overall shift register value (Koopman and Chakravarty, 2004). The initial value of the LFSR is called the seed, because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, the register has a finite number of possible states; it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits, which appears random and, which has a very long cycle (Martin and Steve, 2011). The aim of the paper is to compare various LFSR implementations in the standard polynomials like USB-5, CRC-16-IBM and CRC-32. The analysis is based upon the product of power and area (P-A).

Selection of Polynomial

The selection of polynomial determines the size and the taps of the shift register.

CRC-5-USB

5-bit CRC polynomial is used for providing error detection for Universal Serial Bus (USB) tokens and by an ITU standard for telecommunication systems (Koopman and Chakravarty, 2004). The USB 5-bit CRC standard, "USB-5," is a hexadecimal value 0x12 = x^5 + x^3 + 1. This polynomial is used by USB to protect data words of length 11 bits. USB-5 is optimal for 11-bit messages, and is nearly optimal for longer data word lengths. It is, however, not necessarily a good choice for data words sized 10 and lower, because it is a full bit of HD worse than the bound.

CRC-16-IBM

The CRC-16-IBM is represented as x^16+x^15+ x^2+1 and its hexadecimal value is 0xC002 (Peter, 2015). All single and double-bit errors can be detected using this type CRC and ensures detection of 99.998% of all possible errors. This level of detection is considered sufficient for data transmission blocks of 4 kilobytes or less.

CRC-32

Accidental data changes can be detected using CRC-32. These polynomials are commonly used in networks and storage devices (Koopman, 2002). The purpose of this algorithm is not only focused to protect against intentionally changes, but also to catch accidental changes like network errors, disks write errors, etc. The emphasis of this algorithm is those more on speed than on security. The hexadecimal value for CRC-32 is 0x82608EDB and its polynomial representation is:

x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+ x^8+x^7+x^5+x^4+x^2+x^1+1.

Based on the above case study, each polynomial has specific problems. One problem is that some polynomials provide very poor error detection capabilities. A secondly even a good polynomial will go wrong when misused for messages of a various lengths.

Table 1 shows the uses and representation of polynomials. Therefore, selection of a good polynomial must not only involve on the size, but also the size of the data word. Moreover, many commonly used polynomials are poorly suited to some other applications. Therefore, suitable polynomial for each application has to be chosen.

Table 1. Uses and representation of polynomial

Name	Uses	Polynomial Representation		
		Normal	Reversed	Reversed Reciprocal
CRC -5-USB	USB token packets	0x05	0x14	0x12
CRC-16-IBM	Bisync, Modbus, USB	0x8005	0xA001	0xC002
CRC-32	HDLC, ANSI X3.66, ITU-T V.42, Ethernet, Serial ATA	0x04C11DB7	0xEDB88320	0x82608EDB

Mehods of LFSR Implementation

CRC

Cyclic redundancy check (CRC) is an error detecting code, which is used to detect correction in the block of transmitted data or stored data. The linear feedback shift register (LFSR) is the generic inexpensive hardware used for the CRC, which assumes serial data input. The used polynomial determines the capability of the error detection. The performance of the polynomial is affected by the data, its length as well as the anticipated error patterns (Martin and Steve, 2011)). Different applications might favor different polynomials.

The register needs to be cleared initially in order to obtain the CRC. After the injection of the message and additional zeros, the specific CRC will be hold by the register. The same procedure can be applied at the receiver end to verify the received message with its appended CRC. The only difference is that the CRC will be shifted into the circuit instead of the zeros. The register finally becomes zero, if no error has been detected. Fig. 1 shows the CRC representation using USB 5 polynomial.

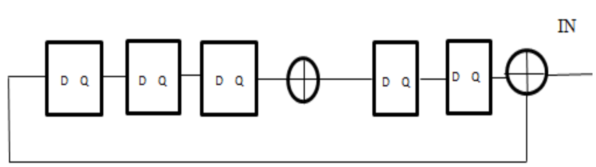

Figure 1: CRC representation using USB 5 polynomial

Galois Model

In the Galois model, bits that are not taps are shifted one position to the right unchanged, when the system is clocked. The taps and output are XOR'd before they are stored in the next position [5]. The new output bit is the input to the next bit. Due to this when the output bit is zero all the bits in the register shift to the right unchanged, and the input bit becomes zero. When the output bit is one, the bits in the tap positions all flip and then the entire register is shifted to the right and the input bit becomes 1. Fig. 2 infers the USB 5 polynomial implementation using Galois's model.

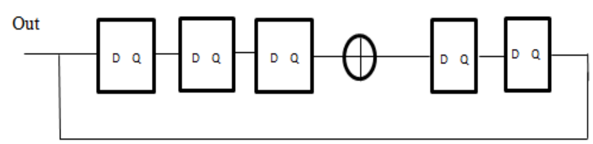

Figure 2: Galois representation using USB 5 polynomial

Fibonacci Model

Fig. 3 shows that the LFSR in the Fibonacci configuration has several been tapped cells (Pritish and Sadawarte, 2015). The contents of the tapped cells are added, and the sum (modulo 2) is returned to the first cell of the shift register for a clock cycle. In Fibonacci model the corresponding connection polynomial is irreducible.

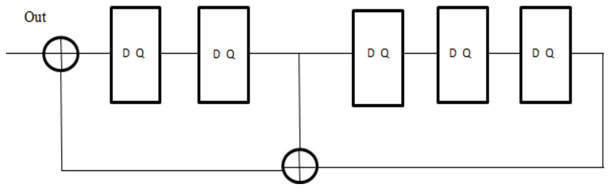

Figure 3: Fibonacci representation using USB 5 polynomial

Additive Scrambler

Additive scramblers are also referred to be as synchronous. It transforms the input data stream by applying a pseudo-random binary sequence (PRBS). It follows modulo-two addition. More often it is generated by LFSR but sometimes a pre-calculated PRBS stored in the Read-only memory is used (Kenneth, 1993). In this type of scrambler, the effective length of the random sequence is limited by the frame length, which is normally much shorter than the period of the PRBS. It is possible to extend the length of the random sequence by adding frame numbers to the frame sync. Fig. 4 infers the Additive scrambler representation using USB 5 polynomial.

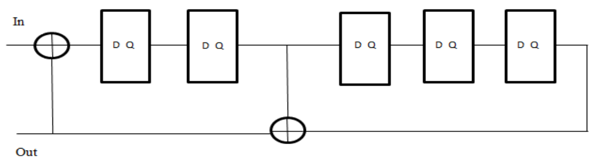

Figure 4: Additive scrambler representation using USB 5 polynomial

Multiplicative Scrambler

Multiplicative scramblers is also known as feed-through, this is because they perform a multiplication of the input signal by the scrambler's transfer function in Z-space. They are discrete linear time-invariant systems (Kenneth, 1993). A multiplicative scrambler is recursive and a multiplicative descrambler is non-recursive. Multiplicative scramblers are also called as self-synchronizing, because it do not requires the frame synchronization. Fig. 5 shows the USB 5 implemented using Multiplicative scrambler.

A single-bit error at the descrambler's input will result into X errors at its output, where X equals the number of the scrambler's feedback taps. This leads to error multiplication during descrambling. Additive scramblers must be reset by the frame sync; if this fails massive error propagation will result as a complete frame cannot be descrambled.

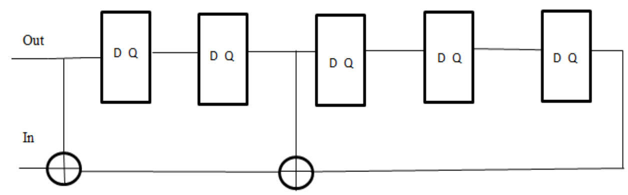

Figure 5: Multiplicative scrambler representation using USB 5 polynomial

Simulation Results and Discussion

To analyze the different LFSR representations, various polynomials are considered. RTL code is verified and synthesized using Synopsys VCS and Cadence RTL compiler targeted to UMC90nm CMOS technology. The design is synthesized for various polynomials ranging from 5 to 32 bits. Figure 6 shows the simulation result for Fibonacci 32 models.

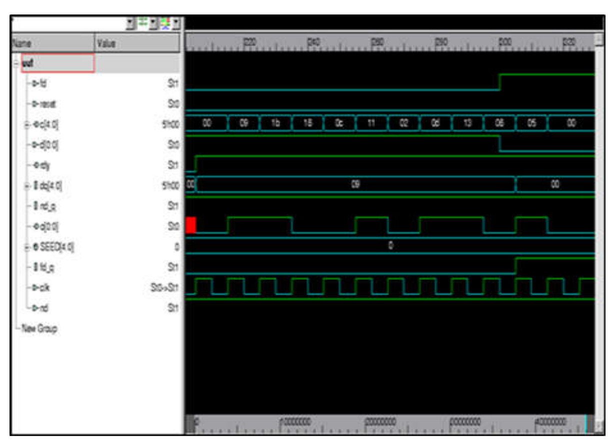

Figure 6: Waveform for Fibonacci 32 using CRC model

After the simulation, various LFSRs have been synthesized using Cadence RTL compiler. The designs are synthesized for constant timing slack of about 4445ps, and the optimized area and power results are obtained.

On comparing with all other models, Fibonacci produces multiple random bits. In addition the Fibonacci configuration can be extended without suffering the number of taps. On the other hand, other models form cannot be extended. Fig. 7 shows the area report for the Fibonacci 32 bit, the total area of about 1206 is obtained and the optimization status produces 1174 total cell area. The Fig. 8 shows the power report for Fibonacci 32bit.

Table 2 shows the area and power comparison report for implementation LFSR model for various 32 bit polynomials.

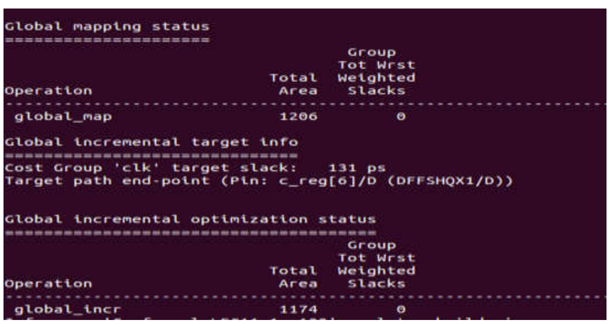

Figure 7: Area report

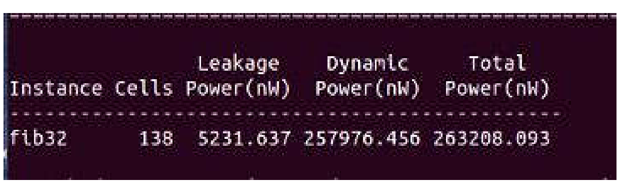

Figure 8: Power report

Table 2. Area and power comparison

LFSR Models (32 Bit)	Area (mm^2)	Power(nW)
CRC	1295	273761.655
Galois	1177	265681.356
Fibonacci	1174	263208.093
Additive Scrambler	1350	283458.98
Multiplicative Scrambler	1357	299785.786

After the successful synthesis the physical design is created using the cadence encounter. The physical design involves the floor planning, routing and generating a GDS II file. Generated netlist from the compiler is imported into cadence encounter. After loading corresponding LEF files and technology libraries, an automated floor plan is done with the suitable ratios.

The core die is surrounded by power rings (VDD and VSS) after the floor planning. Furthermore the horizontal and vertical power stripes across the die are given. Now the design macros are placed across the die so that optimum design is achieved.

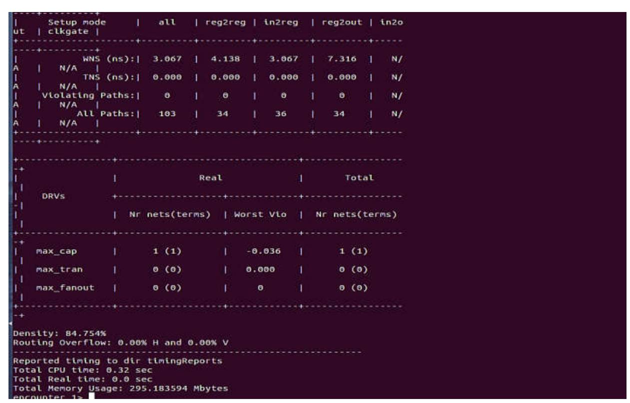

Figure 9: Pre-CTS Timing report

Further, the clock tree synthesis (CTS) is done to minimize skew and insertion delay. Pre-CTS and Post-CTS for both setup and hold mode was carried out. Additionaly optimization is carried out in case of negative slack. Figure 9 and 10 shows the timing report for Pre-CTS and Post-CTS in setup mode.

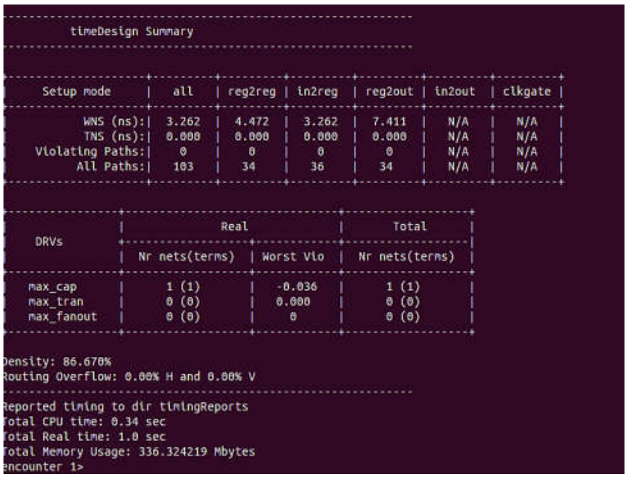

Figure 10: Post-CTS Timing report

Once the clock tree synthesis is done the die is routed in optimum fashion. In Cadence Encounter, permanent routing is done by Nano-Route. Special routing and nano routing are carried out with different metal layers. Figure 11 and 12 show report for nano routing.

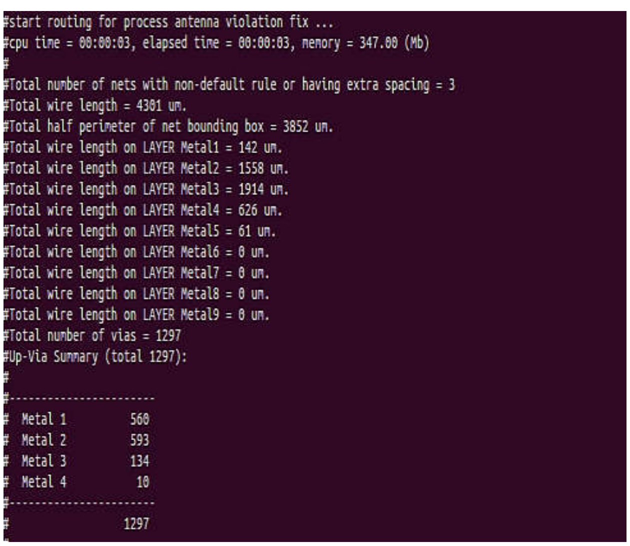

Figure 11: Report for Nano routing

Fig.13 represents the physical view for 32 bit Fibonacci model. Fibonacci model is implemented using External LFSR with several tapped cells whereas in Galois model, the LFSR implementation is based on internal LFSR (Burton, 1999). However, Galois's model also produces reduced power and area for USB-5 and CRC-16-IBM.

On comparing the area and power of all the LFSR implementations, the basic CRC model produces the increased power and area for all the standard polynomials (Vikas and Pradeep, 2013). This is due the fact that, it grows linearly with a higher scaling factor. Moreover, the additive scramblers have worse randomness compared to multiplicative scramblers when the length is short. The scrambler is an additive type of scrambler in contrast to a multiplicative type of scrambler. However, this is typically not a concern for the lengths of data that storage systems typically deal with (*e.g.,* 2K, 4K, etc.). And unlike multiplicative scramblers, additive scramblers can be implemented in parallel but increase in area and power compared to Fibonacci and Galois's model.

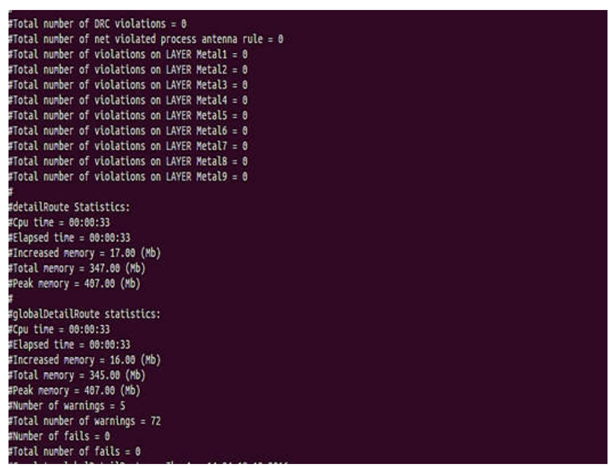

Figure 12: Routing report

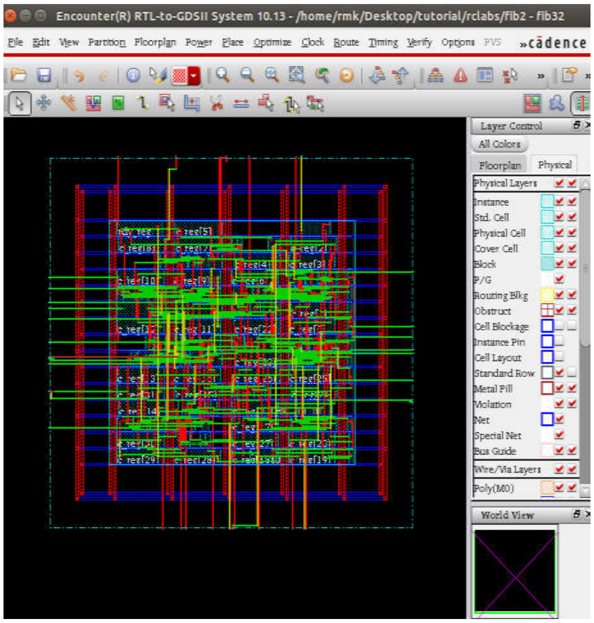

Figure13: Physical view of Fibonacci 32 bit

Conclusion

Based on the whole performance, Fibonacci produces significant power and area with the constant timing slack of 4445ps. When compared to CRC, Galois, Additive scrambler and Multiplicative scrambler, Fibonacci model is more efficient. Moreover, the Fibonacci configuration can be extended without suffering the number of taps. On the other hand, other models cannot be extended.

In general, Galois's model offers more efficiency than Fibonacci form if it handles on LFSR with many taps. In each case, Fibonacci representation is simpler, especially with regard to the computation of the initial loading of the register. Moreover, in all standard polynomial Fibonacci circuitry produces less power and area and also produces multiple random bits. The implemented model can be used in error detection and error correction techniques to prevent from Single event upset (SEU), which is caused due to radiation into the environment.

Acknowledgement

Thanks to the College of Electrical and Mechanical Engineering, AASTU for extending library support and to validate our idea successfully.

Conflict of Interest

The authors declared that there is no conflict of interst regading to this paper.

References

Burton R. (1999). Lecture notes on Cryptography. http://cs.miami.edu/home/burt/learning/Csc599.092/docs/Cryptolecture3fullpage.pdf

Kenneth S. (1993). Scrambler/descrambler system for data transmission. U.S. Patent 1-5

Koopman P. (2002). 32 bit cyclic code redundancy code for internet application. Proceedings of the international conference on Dependable Computer and Networks,459-472.

Koopman P. and Chakravarty T. (2004). Cyclic redundancy code (CRC) polynomial selection for embedded networks. Proceedings of Int. Conf. Depend. Syst. Netw. 145–154.

Martin G. and Steve B.F. (2011). A Novel Programmable Parallel CRC Circuit," IEEE Trans. *Very Large Scale Integr. Syst., 19(10):1898-1902.*

Peter S. (2015). Cyclic Redundancy Check. http://www.stallinga.org/AcadActiv/Lectures/SRT/Exercises/CRC%20(Wikipedia).pdf

Pritish A.D. and Sadawarte Y.A. (2015). Pseudo-Random Number Generation by Fibonacci and Galois LFSR Implemented on FPGA. Int. J. Compu. Applic. (1) 1-3

Vikas S. and Pradeep K. (2013). Power Reduction and Speed Augmentation in LFSR for Improved Sequence Generator using Scaling Principle and Transistor Stacking, *IJARCSEE, 2(1):66-69*